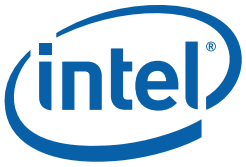

Intel® MPI Library for Linux* OS

Getting Started Guide

The Intel® MPI Library is a multi-fabric message passing library that implements the Message Passing Interface, v2 (MPI-2) specification. Use it to switch between interconnection fabrics without re-linking.

This *Getting Started Guide* explains how to use the Intel® MPI Library to compile and run a simple MPI program. This guide also includes basic usage examples and troubleshooting tips.

To quickly start using the Intel® MPI Library, print this short guide and walk through the example provided.

Contents

Disclaimer and Legal Notices

1 About this Document

The *Intel® MPI Library for Linux* OS Getting Started Guide* contains information on the following subjects:

- First steps using the Intel® MPI Library
- Troubleshooting outlines first-aid troubleshooting actions

1.1 Intended Audience

This *Getting Started Guide* is intended for first time users.

1.2 Using Doc Type Field

This *Getting Started Guide* contains the following sections:

Document Organization

Section	Description
Section 1 About this Document	Section 1 introduces this document
Section 2 Using the Intel® MPI Library	Section 2 describes how to use the Intel® MPI Library
Section 3 Troubleshooting	Section 3 outlines first-aid troubleshooting actions
Section 4 Next Steps	Section 4 provides links to further resources

1.3 Conventions and Symbols

The following conventions are used in this document.

Table 1.3-1 Conventions and Symbols used in this Document

This type style	Document or product names
This type style	Hyperlinks
`This type style`	Commands, arguments, options, file names
`THIS_TYPE_STYLE`	Environment variables
`<this type style>`	Placeholders for actual values
`[ items ]`	Optional items
`{ item | item }`	Selectable items separated by vertical bar(s)
(SDK only)	For Software Development Kit (SDK) users only

1.4 Related Information

To get more information about the Intel® MPI Library, see the following resources:

Product Web Site

Intel® MPI Library Support

Intel® Cluster Tools Products

Intel® Software Development Products

2 Using the Intel® MPI Library

2.1 Usage Model

Using the Intel® MPI Library involves the following steps:

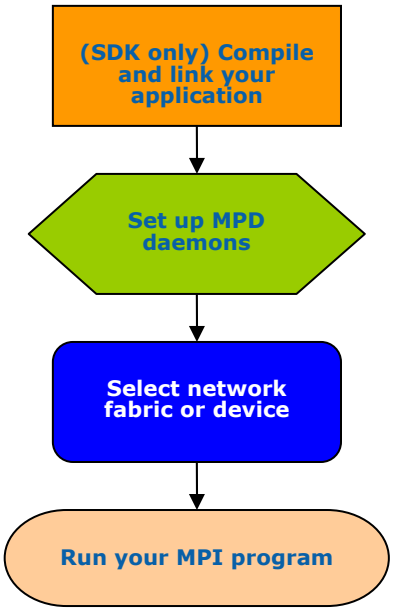

Figure 1: Flowchart representing the usage model for working with Intel MPI Library.

2.2 Before You Begin

Before using the Intel® MPI Library, ensure that the library, scripts, and utility applications are installed. See the product *Intel® MPI Library for Linux* OS Installation Guide* for installation instructions.

2.3 Quick Start

1. Source the `mpivars.[c]sh` script to get proper environment settings included with the Intel® MPI Library. It is located in the `<installdir>/bin` directory, or, for the Intel® 64 architecture in 64-bit mode, the `<installdir>/bin64` directory.
2. Create an `mpd.hosts` text file that lists the nodes in the cluster using one host name per line.
3. **(SDK only)** Make sure you have a compiler in your `PATH`. For example, running the `which` command on the desired compiler returns a path to the executable:

   ```
   $ which icc
   /opt/intel/cc/11.0/bin/icc
   ```
4. **(SDK only)** Compile a test program using the appropriate compiler driver. For example:

```
$ mpiicc -o test <installdir>/test/test.c
```

5. Execute the test using the `mpirun` command.

```
$ mpirun -r ssh -f mpd.hosts -n <# of processes> ./test
```

6. See the rest of this document and the *Intel® MPI Library Reference Manual* for more details.

2.4 Compiling and Linking

(SDK only)

To compile and link an MPI program with the Intel MPI Library:

1. Ensure that the underlying compiler and related software appear in your `PATH`.

   If you use Intel® compilers, ensure that the compiler library directories appear in the `LD_LIBRARY_PATH` environment variable.
   For example, for Intel® C++ Compiler version 11.0 and Intel® Fortran Compiler version 11.0, execute the appropriate setup scripts:

   `/opt/intel/cc/11.0/bin/iccvars.[c]sh`, and

   `/opt/intel/fc/11.0/bin/ifortvars.[c]sh`

2. Compile your MPI program via the appropriate `mpi` command.
   For example, use the `mpicc` command to compile C code using the GNU* C compiler as follows:

   ```
   $ mpicc <installdir>/test/test.c
   ```

   where *<installdir>* is a full path to the installed package.

   All supported compilers have equivalent commands that use the prefix `mpi` on the standard compiler command. For example, the Intel MPI Library command for the Intel® Fortran Compiler (`ifort`) is `mpiifort`.

2.5 Setting up MPD Daemons

The Intel MPI Library uses a Multi-Purpose Daemon (MPD) job startup mechanism. In order to run programs compiled with the `mpiicc` (or related) commands, set up the MPD daemons first.

Always start and maintain your own set of MPD daemons, as opposed to having the system administrator start up the MPD daemons once for use by all users on the system. This setup enhances system security and gives you flexibility in controlling your execution environment.

To set up the MPD daemons:

1. Set up the environment variables with appropriate values and directories. For example, in the `.cshrc` or `.bashrc` files.

   - Ensure that the `PATH` variable includes the *<installdir>*`/bin` directory, or, for the Intel® 64 architecture in 64-bit mode, the *<installdir>*`/bin64` directory. Use the `mpivars.[c]sh` scripts included with the Intel MPI Library to set up this variable.
   - Ensure that the `PATH` variable includes the directory for Python* version 2.2 or higher.
   - **(SDK only)** If you are using the Intel® compilers, ensure that the `LD_LIBRARY_PATH` variable contains the directories for the compiler library. Set this variable by using the `{icc, ifort}*vars.[c]sh` scripts included with the compiler.
   - Set any additional environment variables that your application uses.

NOTE: Steps 2 through 4 are optional. They are automatically completed when using the `mpiexec` or `mpirun` commands.

2. Create a `$HOME/.mpd.conf` file. To set up your MPD password, enter the following into this file:
   `secretword=`*<mpd secret word>*

Do not use a Linux* login password. The arbitrary <mpd secret word> string only controls access to the MPD daemons by various cluster users.

3. Set protection on the $HOME/.mpd.conf file using the chmod command so that only you have read and write privileges:

```
$ chmod 600 $HOME/.mpd.conf
```

4. Verify that you can observe the PATH settings and .mpd.conf contents through rsh on all nodes of the cluster. For example, use the following commands with each <node> in the cluster:

```
$ rsh <node> env
$ rsh <node> cat $HOME/.mpd.conf
```

Make sure that every node, rather than only one of them, can connect to any other node.

You can use the provided sshconnectivity expect script. The script accepts as an argument a file containing the list of all nodes on your system, one host name per line:

```
$ sshconnectivity.exp machines.LINUX
```

If your cluster uses ssh instead of rsh, look into the Notes section below.

5. Create an mpd.hosts text file that lists the nodes in the cluster using one host name per line.

For example:

```
$ cat > mpd.hosts
node1
node2
…
<ctrl>D
```

6. Shut down the eventual MPD daemons using the mpdallexit command:

```
$ mpdallexit
```

7. Use the mpdboot command to start up the MPD daemons:

```
$ mpdboot -n <#nodes>
```

The file $PWD/mpd.hosts will be used by default if it is present. If there is no host file, the mpdboot command without any parameters will start one MPD daemon on the local machine.

8. Use the mpdtrace command to determine the status of the MPD daemons:

```
$ mpdtrace
```

The output should be a list of nodes that are currently running MPD daemons. This should match the list of nodes in the mpd.hosts file, allowing for name resolution.

NOTE: If your cluster uses ssh instead of rsh, make sure that every node can connect to any other node via ssh without a password. Look into your system manuals for details.

NOTE: If your cluster uses ssh instead of rsh, add the -r ssh or --rsh=ssh option to the mpdboot invocation string.

2.6 Selecting a Network Fabric

The Intel® MPI Library dynamically selects the most appropriate fabrics for communication between MPI processes. To select a specific fabric combination, set the I_MPI_DEVICE environment variable to one of the following values:

I_MPI_DEVICE values	Supported fabric
sock	TCP/Ethernet*/sockets
shm	Shared memory only (no sockets)
ssm	TCP + shared memory (for SMP clusters connected via Ethernet*)
rdma[:<provider>]	InfiniBand*, Myrinet* (via specified DAPL* provider)
rdssm[:<provider>]	TCP + shared memory + DAPL* (for SMP clusters connected via RDMA-capable fabrics)

Ensure that the selected fabric is available. For example, use `shm` only when all the processes can communicate with each other via shared memory. Use `rdma` only when all processes can communicate with each other via a single DAPL provider.

2.7 Running an MPI Program

2.7.1 Running an MPI program with mpiexec

To launch programs linked with the Intel MPI Library, use the `mpiexec` command:

```
$ mpiexec -n <# of processes> ./myprog
```

Use the `-n` option to set the number of processes. This is the only obligatory option for the `mpiexec` command.

If you are using a network fabric different than the default fabric, use the `-genv` option to specify a value to be assigned to the `I_MPI_DEVICE` variable.

For example, to run an MPI program using the `shm` fabric, type in the following command:

```
$ mpiexec -genv I_MPI_DEVICE shm -n <# of processes> ./myprog
```

For the `rdma` capable fabric, use the following command:

```
$ mpiexec -genv I_MPI_DEVICE rdma -n <# of processes> ./myprog
```

You can select any supported device. For more information, see *Selecting a Network Fabric* above.

If you successfully ran your application using the Intel MPI Library, you can move your application from one cluster to another and use different fabrics between the nodes without re-linking. If you encounter problems, see *Troubleshooting* for possible solutions.

2.7.2 Running an MPI program with mpirun

An alternative way to run parallel applications using the Intel MPI Library is to type in the following command:

```
$ mpirun -n <# of processes> ./myprog
```

This command invokes `mpdboot`, `mpiexec`, and `mpdallexit` commands. `mpirun` shuts the MPD daemons down as soon as the program execution is completed. Use this command when you do not need the MPD daemons after the program run. Using `mpirun` is the recommended practice when using a resource manager, such as PBS Pro* or LSF*.

3 Troubleshooting

Use the following sections to troubleshoot problems with installation, setup, and execution of applications using the Intel® MPI Library.

3.1 Testing Installation

To ensure that the Intel® MPI Library is installed and functioning correctly, complete the general testing, compile and run a test program.

To test the installation:

1. Verify that you have Python* version 2.2 or higher in your `PATH`:
   ```
   $ rsh <nodename> python -V
   ```
   If this command returns an error message or a value lower than 2.2, install Python* version 2.2 or higher, and make sure that you have it in your `PATH`.
2. Check for the presence of a Python* XML module such as **python-xml** or **libxml2-python**:
   ```
   $ rpm -qa | grep python-xml
   $ rpm -qa | grep libxml2-python
   ```
   Install the missing module if the output does not include the name **python-xml** or **libxml2-python** and a version number.
3. Check for the presence of an XML parser such as **expat** or **pyxml**:
   ```
   $ rpm -qa | grep expat
   $ rpm -qa | grep pyxml
   ```
   Install the missing module if the output does not include the name **expat** or **pyxml** and a version number.
4. Verify that `<installdir>/bin` (`<installdir>/bin64` for the Intel® 64 architecture in 64-bit mode) is in your `PATH`:
   ```
   $ rsh <nodename> which mpiexec
   ```
   You should see the correct path for each node you test.
   (SDK only) If you use the Intel® compilers, verify that the appropriate directories are included in the `PATH` and `LD_LIBRARY_PATH` environment variables
   ```
   $ mpiexec -n <# of processes> env | grep PATH
   ```
   You should start an `mpd` ring before executing the `mpiexec` command. You should see the correct directories for these path variables for each node you test. If not, call the appropriate `{icc, ifort}*vars.[c]sh` scripts. For example, for the Intel® C++ Compiler version 11.0 use the following source command:
   ```
   $ . /opt/intel_cc_11.0/bin/iccvars.sh
   ```
5. In some unusual circumstances, you may need to include the `<installdir>/lib` directory (`<installdir>/lib64` for the Intel® 64 architecture in 64-bit mode) in your `LD_LIBRARY_PATH`. To verify your `LD_LIBRARY_PATH` settings, use the command:
   ```
   $ mpiexec -n <# of processes> env | grep LD_LIBRARY_PATH
   ```

3.2 Troubleshooting MPD Setup

Check if it is possible to run the `mpd` command on the local machine. Do the following:

```
$ mpd &
$ mpdtrace
```

```
$ mpdallexit
```

The output of `mpdtrace` should show the hostname of the machine you are running on. If this is not the case, or if you cannot start up the MPD, check that the installation was correct and the environment was set up properly.

The next troubleshooting steps assume that the MPD daemons are set up and running. To start your diagnosis, verify that the MPD daemons are running on all expected nodes using:

```
$ mpdtrace
```

The output lists all MPD daemons running or indicates an error. If some desired nodes are missing from the output list of `mpdtrace`, do the following:

1. Try to restart the MPD daemons using the following commands:
   - Kill all running MPD daemons:
     ```
     $ mpdallexit
     ```
   - For each node, ensure all daemons were killed:
     ```
     $ rsh <nodename> ps -ael | grep python
     $ rsh <nodename> kill -9 <remaining python processes>
     ```
   - Reboot the MPD daemons. Be sure to use the appropriate configuration options and host file:
     ```
     $ mpdboot [<options>]
     ```
   - Confirm that all expected MPD daemons are now running:
     ```
     $ mpdtrace
     ```
2. If the output of the `mpdtrace` command is still not indicating that all expected MPD daemons are running, follow the next steps:
   - Kill and restart the MPD daemons as described in step 1, adding the debug and verbose options to the `mpdboot` command:
     ```
     $ mpdboot –d –v [<options>]
     ```
     Find the `rsh` commands in the output. For example:
     ```
     launch cmd= rsh  -n <nodename> '<installdir>/bin/mpd \
     -h <nodename> -p <port-number> --ncpus=<ncpus> -e -d'
     ```
   - Copy and paste the line of the output from the `rsh` command up to the end of the line. For example:
     ```
     $ rsh -n <nodename> '<installdir>/bin/mpd \
     -h <nodename> -p <port-number> --ncpus=<ncpus> -e -d'
     ```
   - Execute the edited `rsh` command. Use the resulting output to diagnose and correct the underlying problem. For example, the most common problems include:
     - Failure of the `rsh` command to contact *<nodename>*.
     - Other failure of the `rsh` command, for example, a system setup problem.
     - The *<installdir>*`/bin/mpd` command could not be found or could not be executed.

3.3 Compiling and Running a Test Program

To compile and run a test program, do the following:

1. **(SDK only)** Compile one of the test programs included with the product release as follows:
   ```
   $ cd <installdir>/test
   $ mpiicc test.c
   ```
2. If you are using InfiniBand*, Myrinet*, or other RDMA-capable network hardware and software, verify that everything is functioning correctly using the testing facilities of the respective network.
3. Run the test program with all available configurations on your cluster.

- Test the `sock` device using:
  `$ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE sock ./myprog`
  You should see one line of output for each rank, as well as debug output indicating the `sock` device is being used.
- Test the `ssm` devices using:

  `$ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE ssm ./ myprog`
  You should see one line of output for each rank, as well as debug output indicating the `ssm` device is being used.
- Test any other fabric devices using:

  `$ mpiexec -n 2 -env I_MPI_DEBUG 2 -env I_MPI_DEVICE <device> ./ myprog`
  where `<device>` can be `shm`, `rdma`, or `rdssm`

For each of the `mpiexec` commands used, you should see one line of output for each rank, as well as debug output indicating which device was being used. The device(s) should agree with the `I_MPI_DEVICE` setting.

NOTE: The `<installdir>/test` directory in the Intel MPI Library Development Kit contains other test programs in addition to `test.c`

4 Next Steps

To get more information about the Intel® MPI Library, explore the following resources:

See the *Intel® MPI Library Release Notes* for updated information on requirements, technical support, and known limitations.

The *Intel® MPI Library Reference Manual* for in-depth knowledge of the product features, commands, options, and environment variables.

For more information see Websites:

Product Web Site

Intel® MPI Library Support

Intel® Cluster Tools Products

Intel® Software Development Products